复旦大学

2001年招收攻读硕士学位研究生入学考试试题

报考专业：通信与信息系统
计算机系统结构
计算机软件与理论
计算机应用技术

考试科目：数据结构与操作系统

（共 8 页）

1. 请阐述下列各对概念间的主要不同之处。（共 10 分）

1)	Kernel-level Thread	和	Lightweight Process.
2)	Deadlock	和	Starvation.
3)	Paging	和	Swapping.
4)	SCAN scheduling	和	C-SCAN scheduling.
5)	Indexed Sequential File	和	Hashed File.

2. 简答题（共 12 分）

1) 请详细列出作业、进程的状态转换图，并注明状态转换的类型原因.
2) 请阐述对于互斥临界区的管理要求.
3) 请阐述 OPEN( )和 CLOSE( )作用.

223

3. 下述算法是解决 Reader and Writer 问题的一种实现方法。请详细阐述这个算法的平等性，说明为什么？ (共 8 分)

```
MONITOR ReaderAndWriters;
VAR
    readers                          : integer;
    someoneiswriting                 : boolean;
    readingallowed, writingallowed   : condition;
PROCEDURE beginreading;
    begin
        if  someoneiswriting or queue(writingallowed)
            then  wait(readallowed)
        readers := readers + 1;
        signal(readingallowed);
    end;
PROCEDURE finishedreading;
    begin
        readers := readers - 1;
        if  readers = 0  then  signal(writingallowed)
    end;
PROCEDURE beginwriting;
    begin
        if  readers > 0  or  someoneiswriting
            then  wait(writingallowed)
        someoneiswriting := true;
    end;
PROCEDURE finishedwriting;
    begin
        someoneiswriting := false;
        if  queue(readingallowed)
            then  signal(readingallowed)
            else  signal(writingallowed)
    end;

begin
    readers          := 0;
    someoneiswriting := faise;
end
```

4. 请使用堆排序所使用的调整算法把存放在数组 A[ ]前 10 个元素中的数据 45，25，15，80，50，75，60，40，35，70 调整成一个堆。（6 分）

5. 如果我们称自右上角到左下角的对角线为次对角线，那么一个 n 阶方阵 a[n][n]的次对角线共有(2n-1)条。现在按照从右上方次对角线到左下方次对角线的次序逐条进行存储，而每条次对角线的元素自左上角到右下角依次存放在一维数组 b[n*n]中。例如：对于四阶方阵 a[4][4]，可按照上面规定的次序存放在一维数组 b[16]中，（见下图）。

1	2	3	4	
5	6	7	8	a[4][4]
9	10	11	12	
13	14	15	16	

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b[i]	1	2	5	3	6	9	4	7	10	13	8	11	14	12	15	16

如果：a[i][j] (0<= i， j <=n-1)存放在 b[k] (0<= k <=n*n-1)中，那么请写出求得 k 的计算公式 k=f(i, j, n)。要求给出推导过程。（10 分）

注 意：在下面三个程序填充题中，要求在每个空格中填上适当内容，每个空格只填一个表达式或一个语句。

6. 在后面所给的程序段 program_1 中，函数 create( )用于建立 n 个结点的链表，函数 print( )用于打印链表，函数 exchange( )以给定的链表 head 的第一个结点的值为标准，把小于此值的结点移到链表的前面，将大于等于此值的结点移到链表的后面。（18 分）

7. 假设二叉树 t 有 n 个结点，我们按前序将树 t 的所有结点存放在数组 data[ ]的前 n 个元素中，值为 data[i]的结点有相应指针 right[i]指向该结点的右子结点存放在 data[ ]中的存放位置，若该结点没有右子结点，则置 right[i]为 -1。我们称二叉树的这种存储方法为按前序且附带一个指向右子结点的指针的顺序存储。在后面的程序段 program_2 中，函数 create( )将二叉树按照上述的顺序存储转换成附加两个分别指向左、右子结点的指针的链接存储。程序中利用一个栈帮助我们对二叉树 t 实现上述的存储转换。 (18 分)

8. 设 t 是一棵二叉树，如果 t 中的非叶子结点都有两棵非空的子树，那么称 t 是一棵完全二叉树。一棵二叉树不能由它的前序和后序唯一确定，但一棵完全二叉树可由它的前序和后序唯一确定。在后面所给的程序段 program_3 中，函数 complete_tree( )就是利用这个结论构造我们所需的完全二叉树。数组 pre[ ]和 pos[ ]的前 n 个元素分别存放完全二叉树的前序和后序。程序中使用栈帮助我们构造所需的完全二叉树。(18 分)

```
/*        program_1        */
typedef struct node { int  data;
          struct   node   *link;
} NODE;
NODE    *create(int n)
{
    NODE    *p;
    if  (n==0)  return      (NULL);
    p = (NODE *)malloc(sizeof(NODE));
    scanf("%d",   &p→data);
    _____(A)_____ ;
    return(p);
}
```

```
void print(NODE  *head)
{
    if   ( head != NULL )
    {    printf( "%5d",  head→data );
         _____(B)_____ ;
    }
}
NODE  *exchange(NODE  *head)
{   NODE   *p, *q, *h = NULL, *r = NULL;
    Int        t;
if  ( head != NULL  &&  head →link != NULL) {
    t = head→data;
    p=head;
    while (p→link != NULL)
    if  ( p→link→data < t)
    {
      q = p→link;
      _____(C)_____ ;
      if  (h==NULL)  h=q;
      else  _____(D)_____ ;
      r = q;
    }
    else  _____(E)_____ ;
    if  (h != NULL)
    {
          _____(F)_____ ;
          head=h;
    }
}
return(head);
}
```

```
/*        program_2        */

#define   MAXN   30
typedef struct node
        {   char data;
            struct node   *lchild, *rchild;
} NODE;
char data[MAXN];
int  right[MAXN];
int  n;
NODE  *t;
NODE  *create_tree(char data[], int right[], int n)
{
    NODE  *stack[MAXN],  *p = NULL;
    int top = 0, i, j;
    i = n;
    while (--i >= 0) {
        p = (NODE *) malloc(sizeof(NODE));
        p→data = data[i];
        if  (_____(A)_____)
            p→lchild = NULL;
        else {
            for  (j = 0;  j < i  &&  _____(B)_____;  j++);
                if  (j < i)
                    p→lchild = NULL;
                else
                    _____(C)_____;
        }
        if  (_____(D)_____)
            p→rchild = NULL;
        else
            _____(E)_____;
            _____(F)_____;
        }
    return(p);
}

/*        t=create_tree(data,right,n);        */
```

```
/*        program_3        */

#define   MAXN  30
typedef struct node {
          char    data;
          struct  node   *lchild, *rchild;
} NODE;
typedef struct snode {
          NODE   *addr;
          int       low,   up;
} SNODE;
SNODE   stack[MAXN];
char      pre[MAXN],   pos[MAXN];
int       top = 0,   n;
NODE   *create_node(char c)
{
     NODE *p;
     p = (NODE *) malloc(sizeof(NODE));
     p→data = c;
     p→lchild = p→rchild = NULL;
     return(p);
}
void   push(NODE *p,   int i,   int j)
{
     stack[top].addr   = p;
     stack[top].low    = i;
     stack[top++].up   = j;
}
```

```
void   pop(NODE **p_p, int *p_i, int *p_j)
{
    SNODE *s;
    *p_p = stack[--top].addr;
    *p_i = stack[top].low;
    *p_j = stack[top].up;
}
int search(char pre[],   int i,   int j,   int x)
{
    int   k;
    for   (k = j;   k >= i   &&   pre[k] != x;   k--);
    return(k);
}

NODE *complete_tree(char pre[], char pos[], int n)
{
    NODE   *t, *p, *q;
    int    i, j, k, m;
    if   (n <= 0)   return   (NULL);
    t = create_node(pos[n - 1]);
    k = n - 1;
    push(t, 1, k);
    while   (_____(A)_____)
    {
        _____(B)_____;
        q=create_node(pos[k]);
        if   (_____(C)_____)   p→rchild=q;
        else   _____(D)_____;
        m = search(pre, i, j, pos[k]);
        if   (m > i)   _____(E)_____;
        if   (m < j)   _____(F)_____;
    }
    return(t);
}
```